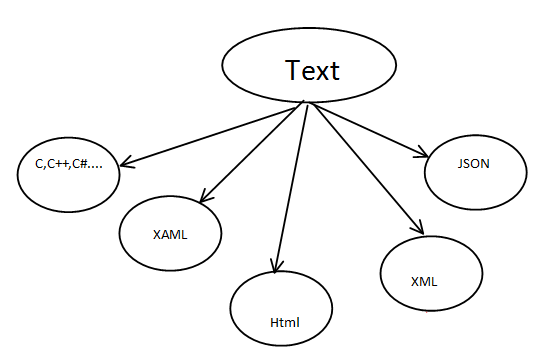
C# 温故而知新：Stream篇（**三**）TextWriter 和 StreamWriter

为何介绍TextWriter?

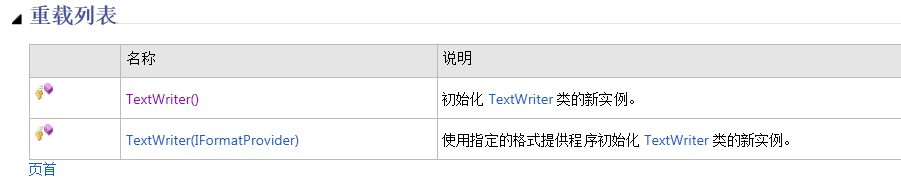
就像上篇讲述的一样，对于重要的基础技术，我们一定要刨根问底，这样在面对将来可能很复杂的业务或技术时才能游刃有余，甚至可以创新出新的解决方案，言归正传，想了解StreamWriter 必须了解其父亲TextWriter的结构和使用方法。那么微软为什么要创建立这个抽象类呢?看下图



的确可以这样理解C,C++ ,XAML,Html一切可以写在文本上写的语言都可以用Text这个词在抽象，（千万别小看记事本，它才是元老啊），聪明的你想到了，今后我们可以自定义一些自己Writer类来实现我们特定的写功能。现在我们只要理解TextWriter是一个抽象的文本写入器，可以在文本上写入我们想要的格式，可以通过微软派生类或着自定义派生类来实现TextWriter的功能。只要你有足够的想象力就能在创新出一个新的派生类。

TextWriter的构造,常用属性和方法

以下是TextWriter构造函数：和所有的抽象类一样，该类不能直接实例化,它有2个构造函数



特别我想说下第二个构造函数，大家发现这个构造有个IFomatProvider 类型的参数， 这个是什么东东？

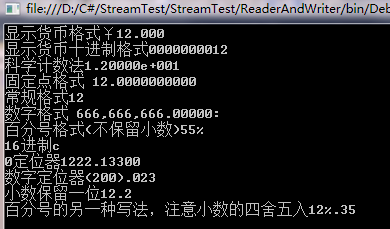
IFormatProvider接口的简单介绍

其实IFormatProvider接口 从字面上就能理解了，一个格式化的提供者

大家记得我们常用的string.format(“{0:P}”,data);么? IFormatProvider在这里被隐式的调用了关于隐式调用的各种方式，用个简单的例子向大家说明下：

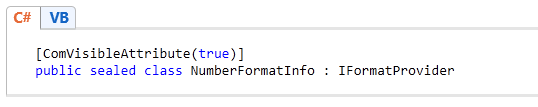
//有关数字格式化隐性使用IFomatProvider的例子  
#if true  
 //货币  
 Console.WriteLine(string.Format("显示货币格式{0:c3}",12));  
 //十进制  
 Console.WriteLine("显示货币十进制格式{0:d10}", 12);  
 //科学计数法  
 Console.WriteLine("科学计数法{0:e5}",12);  
 //固定点格式  
 Console.WriteLine("固定点格式 {0:f10}",12);  
 //常规格式  
 Console.WriteLine("常规格式{0:g10}",12);  
 //数字格式(用分号隔开)  
 Console.WriteLine("数字格式 {0:n5}:",666666666);  
 //百分号格式  
 Console.WriteLine("百分号格式(不保留小数){0:p0}",0.55);  
 //16进制  
 Console.WriteLine("16进制{0:x0}", 12);  
 // 0定位器 此示例保留5位小数，如果小数部分小于5位，用0填充  
 Console.WriteLine("0定位器{0:000.00000}",1222.133);  
 //数字定位器  
 Console.WriteLine("数字定位器{0:(#).###}", 0200.0233000);  
 //小数  
 Console.WriteLine("小数保留一位{0:0.0}", 12.222);  
 //百分号的另一种写法，注意小数的四舍五入  
 Console.WriteLine("百分号的另一种写法，注意小数的四舍五入{0:0%.00}", 0.12345);  
 Console.WriteLine("\n\n");  
 #endif

输出结果：



也就是说IFormatProvider 提供了一个格式化的工具。

让我们通过NumberFormatInfo类来温故下：



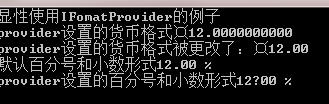
这个密封类实现了IFormatProvider接口，主要实现了一个数字格式化的类，下面是一些规定的格式说明符：

|  |  |
| --- | --- |
| c、C | 货币格式。关联的属性包括： |
| d、D | 十进制格式。 |
| e、E | 科学计数（指数）格式。 |
| f、F | 固定点格式。 |
| g、G | 常规格式。 |
| n、N | 数字格式。 |
| p、P | 百分比格式。 |

                  让我们用简单易懂的代码来实现下NumberFormatInfo 如何使用:

#if true  
 //显性使用IFomatProvider  
 Console.WriteLine("显性使用IFomatProvider的例子");  
 //实例化numberFomatProvider对象  
 NumberFormatInfo numberFomatProvider = new NumberFormatInfo();  
 //设置该provider对于货币小数的显示长度  
 numberFomatProvider.CurrencyDecimalDigits = 10;  
 //注意：我们可以使用C+数字形式来改变provider提供的格式  
 Console.WriteLine(string.Format(numberFomatProvider, "provider设置的货币格式{0:C}", 12));  
 Console.WriteLine(string.Format(numberFomatProvider, "provider设置的货币格式被更改了：{0:C2}", 12));  
 Console.WriteLine(string.Format(numberFomatProvider, "默认百分号和小数形式{0:p2}", 0.12));  
 //将小数 “.”换成"?"   
 numberFomatProvider.PercentDecimalSeparator = "?";  
 Console.WriteLine(string.Format(numberFomatProvider, "provider设置的百分号和小数形式{0:p2}", 0.12));  
 Console.ReadLine();  
#endif

                 输出结果：



                 正如上述代码所表示的，IFormatProvider提供用于检索控制格式化的对象的机制。我们甚至可以自定义provider类来实现特殊的字符串格式化，关于这个重要的知识点我会在另一篇文章中详细介绍并且自定义一个简单的FormatInfo类

言归正传让我们理解下TextWriter的几个重要属性

\*1：Encoding: 可以获得当前TextWriter的Encoding

\*2：FormatProvider: 可以获得当前TextWriter的IFormatProvider

\*3：NewLine: 每当调用WriteLine()方法时，行结束符字符串都会写入到文本流中,该属性就是读取该结束符字符串

 方法：

\*1：Close():关闭TextWriter并且释放TextWriter的资源

\*2：Dispose(): 释放TextWriter所占有的所有资源(和StreamReader相似，一旦TextWriter被释放，它所占有的资源例如Stream会一并释放)

\*3：Flush(): 和Stream类中一样，将缓冲区所有数据立刻写入文件（基础设备）

\*4：Write()方法的重载（这个方法重载太多了，所以这里就不全写出了，大家可以参考最后一个例子的打印结果）

\*5：WriteLine()方法的重载：和Write()方法相比区别在于每个重载执行完毕之后会附加写入一个换行符

如何理解StreamWriter?

首先我们先了解下StreamWriter的概念：实现一个 [TextWriter](http://msdn.microsoft.com/zh-cn/library/system.io.textwriter(v=vs.80).aspx)，使其以一种特定的编码向流中写入字符。那会有很多朋友会疑惑，StreamWriter和TextWriter有什么区别？其实从名字定义我们便可区分了，TextWriter分别是对连续字符系列处理的编写器，而StreamWriter通过特定的编码和流的方式对数据进行处理的编写器StreamWriter的构造函数

\*1：public StreamWriter(string path);  参数path表示文件所在的位置

\*2：public StreamWriter(Stream stream, Encoding encoding);  参数Stream 表示可以接受stream的任何子类或派生类，Encoding表示让StreamWriter 在写操作时使用该encoding进行编码操作

\*3：public StreamWriter(string path, bool append); 第二个append参数非常重要，当append参数为true时，StreamWriter会通过path去找当前文件是否存在，如果存在则进行append或overwrite的操作，否则创建新的文件

\*4：public StreamWriter(Stream stream, Encoding encoding, int bufferSize);bufferSize参数设置当前StreamWriter的缓冲区的大小

StreamWriter的属性

StreamWriter的方法大多都继承了TextWriter 这里就不在重复叙述了，这里就简单介绍下StreamWriter独有的属性

\*1：[AutoFlush](http://msdn.microsoft.com/zh-cn/library/system.io.streamwriter.autoflush(v=vs.80).aspx): 这个值来指示每次使用streamWriter.Write()方法后直接将缓冲区的数据写入文件（基础流）

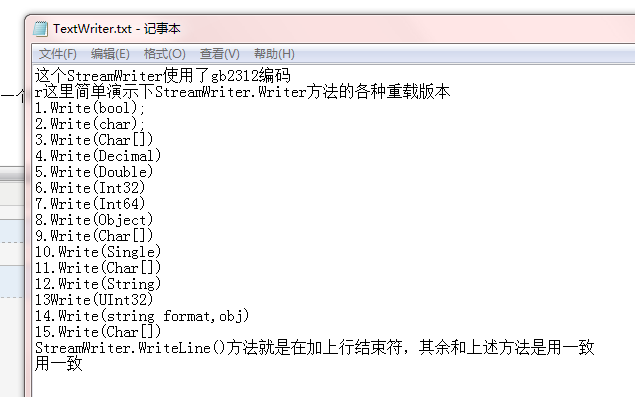
\*2：[BaseStream](http://msdn.microsoft.com/zh-cn/library/system.io.streamwriter.basestream(v=vs.80).aspx): 和StreamReader相似可以取出当前的Stream对象加以处理

StreamWriter示例

const string txtFilePath = "D:\\TextWriter.txt";  
 static void Main(string[] args){  
 NumberFormatInfo numberFomatProvider = new NumberFormatInfo();  
 //将小数 “.”换成"?"   
 numberFomatProvider.PercentDecimalSeparator = "?";  
 StreamWriterTest test = new StreamWriterTest(Encoding.Default, txtFilePath, numberFomatProvider);  
 //StreamWriter  
 test.WriteSomthingToFile();  
 //TextWriter  
 test.WriteSomthingToFileByUsingTextWriter();  
 Console.ReadLine();  
 }  
 }  
 /// <summary>TextWriter和StreamWriter的举例</summary>  
 public class StreamWriterTest {  
 /// <summary>编码</summary>  
 private Encoding \_encoding;  
 /// <summary>IFomatProvider</summary>  
 private IFormatProvider \_provider;  
 /// <summary>文件路径</summary>  
 private string \_textFilePath;

public StreamWriterTest(Encoding encoding, string textFilePath): this(encoding, textFilePath, null){}  
 public StreamWriterTest(Encoding encoding, string textFilePath, IFormatProvider provider){  
 this.\_encoding = encoding;  
 this.\_textFilePath = textFilePath;  
 this.\_provider = provider;  
 }  
 /// <summary>/// 我们可以通过FileStream 或者 文件路径直接对该文件进行写操作/// </summary>  
 public void WriteSomthingToFile(){  
 //获取FileStream  
 using (FileStream stream = File.OpenWrite(\_textFilePath)){  
 //获取StreamWriter  
 using (StreamWriter writer = new StreamWriter(stream, this.\_encoding)){  
 this.WriteSomthingToFile(writer);  
 }  
  
 //也可以通过文件路径和设置bool append，编码和缓冲区来构建一个StreamWriter对象  
 using (StreamWriter writer = new StreamWriter(\_textFilePath, true, this.\_encoding, 20)){  
 this.WriteSomthingToFile(writer);  
 }  
 }  
 }  
 /// <summary>/// 具体写入文件的逻辑/// </summary>  
 /// <param name="writer">StreamWriter对象</param>  
 public void WriteSomthingToFile(StreamWriter writer){  
 //需要写入的数据  
 string[] writeMethodOverloadType ={  
 "1.Write(bool);",  
 "2.Write(char);",  
 "3.Write(Char[])",  
 "4.Write(Decimal)",  
 "5.Write(Double)",  
 "6.Write(Int32)",  
 "7.Write(Int64)",  
 "8.Write(Object)",  
 "9.Write(Char[])",  
 "10.Write(Single)",  
 "11.Write(Char[])",  
 "12.Write(String)",  
 "13Write(UInt32)",  
 "14.Write(string format,obj)",  
 "15.Write(Char[])"  
 };  
 //定义writer的AutoFlush属性，如果定义了该属性，就不必使用writer.Flush方法  
 writer.AutoFlush = true;  
 writer.WriteLine("这个StreamWriter使用了{0}编码", writer.Encoding.HeaderName);  
 //这里重新定位流的位置会导致一系列的问题  
 //writer.BaseStream.Seek(1, SeekOrigin.Current);  
 writer.WriteLine("这里简单演示下StreamWriter.Writer方法的各种重载版本");  
  
 writeMethodOverloadType.ToList().ForEach((name) => { writer.WriteLine(name); });  
 writer.WriteLine("StreamWriter.WriteLine()方法就是在加上行结束符，其余和上述方法是用一致");  
 //writer.Flush();  
 writer.Close();  
 }   
 public void WriteSomthingToFileByUsingTextWriter(){  
 using (TextWriter writer = new StringWriter(\_provider)){  
 writer.WriteLine("这里简单介绍下TextWriter 怎么使用用户设置的IFomatProvider，假设用户设置了NumberFormatInfoz.PercentDecimalSeparator属性");  
 writer.WriteLine("看下区别吧 {0:p10}", 0.12);  
 Console.WriteLine(writer.ToString());  
 writer.Flush();  
 writer.Close();  
 }  
 }  
 }

StreamWriter输出结果：



TextWriter 输出结果

